To appear in *Proceedings of the Twenty-sixth Annual Computer Software and Applications Conference*, IEEE (Oxford, England, August 2002).

On the Response Policy of Software Decoys: Conducting Software-based Deception in the Cyber Battlespace

James Bret Michael
Department of Computer Science
Naval Postgraduate School
Monterey, California 93943-5118
bmichael@nps.navy.mil

Abstract

Intelligent software decoys employ deception techniques to maintain the interaction between themselves and calling processes or threads that violate the contracts of the software components that the decoys defend. The software decoy's goal is to learn about the nature of such interactions before either terminating the interaction or treating the calling process or thread as a cyber combatant. Software components can be wrapped at any level of abstraction, from web applets to operating system calls. It is foreseeable that the decoying actions, termination of interaction, or counterattack by the decoy could in some way violate the law of armed conflict. In this paper we examine the response policy of software decoys in terms of discrimination, necessity, proportionality, and chivalry on the cyber battlefield.

1. Introduction

There has been a shift in the paradigm for computing and communications within the U.S. Navy from platform- to network-centric warfare [1], with the goal of maintaining information superiority over adversaries by utilizing a cooperative engagement grid that provides for the timely exchange of information between U.S., allied, and coalition forces about the ever-changing battlespace. A major focus of the move to network-centric warfare has been on integrating heterogeneous globally distributed systems into an interoperable high-performance information grid.

The next shift in paradigm will likely result in what could be termed "knowledge-centric warfare" [2]. In this paradigm, the focus will shift from that of interoperability between nodes at the physical, data, and information levels to one of leveraging semantic interoperability among the nodes of the cooperative engagement grid, with the goal of improving the effectiveness of forces to share knowledge with each other in a timely manner about the competitive space elements of the battlespace.

Automation of the exchange of knowledge among nodes in the cooperative engagement grid might be spurred on by utilizing the concept of a "semantic web," as introduced by Berners-Lee, Hendler, and Lassila [3]. In the semantic web, knowledge of how to interpret data and information is embedded in the web itself so that software agents can automatically interpret the data and information contained in web pages. For example, one could envision software agents both automatically locating and interpreting the contents of web pages in order to generate and then deliver the latest strategic and tactical information to military commanders.

However, Thuraisingham, Hughes, and Allen [4] contend that for the semantic web to be effective "we need to ensure that the data and information on the web is timely, accurate, and precise." The semantic web, if used for automatically generating and disseminating situational-awareness reports about the battlespace, could become a tempting target for an adversary to attack. An adversary might try to gain a competitive advantage over its opponent by reducing in some way the timeliness, accuracy, or precision of data and information utilized by the software agents on the targeted cooperative engagement grid.

Consider the following scenario. An adversary compromises an opponent's software agent; the software agent is now a *rogue agent* whose goals are managed by the adversary. In order to mislead the rival's forces, the adversary might task the rogue agent with modifying XML tags used by the adversary's foe to specify relationships between the location and strength of the adversary's forces. Other rogue agents might be given different goals, such as to perform distributed denial-of-service attacks on the targeted web or change the behavior of the applets associated with the content of web pages.

The foregoing scenario highlights the need for developing security policy and mechanisms to protect the semantic web from sabotage. In this paper we discuss the role that intelligent software decoys can play in defending semantic webs and other computational structures comprising a cooperative information grid, with particular emphasis on the constraints that would need to be placed on the automatic responses of software decoys to sus-

pected attempts by rogue software agents to sabotage a cooperative information grid.

2. Intelligent Software Decoys

Michael *et al*. [5] introduced the notion of *cyber Aikido*, in which a software component tolerates violations of its contract (*e.g.*, receiving the wrong sequence and type of arguments) in order to learn about the nature of interactions with calling processes or threads; such a component is referred to as an *intelligent software decoy* [6], while the calling process or thread that violates the contract is termed a *cyber opponent*. Here we use the term "software component" in the context of component architectures as defined by Szyperski [7]. A *software component* is a set of *atomic components*, with each atomic component consisting of a *module* (*i.e*., set of classes, procedures, or functions) and a set of *resources* (*i.e*., "a 'frozen' collection of typed items"). A *software contract* [8] is a specification of the obligations and benefits between the component and the calling process or thread. In a distributed system, the contract is a component's public interface that is advertised to other components.

Intelligent software decoys attempt to neutralize the opponent by applying a cyber version of what is known in the martial arts as the Unified Power of Attack [9], which consists of the following sequence of actions: reducing or eliminating the will of the opponent to attack, changing the proximity of attack, and as a last resort reducing or eliminating the ability of the opponent to attack.

The software decoy conducts information operations, employing deception strategies to reduce or eliminate the opponent's will to violate the contract (*e.g.*, by inserting delays into responses to method calls made by the opponent), and to change the proximity of the software component with respect to the opponent (*e.g.*, directing the attention of the opponent to a honeypot). The software decoy at some point, such as after recognizing a known pattern of intrusion (*e.g*., a sequence of operations to effect a buffer-overflow attack in application- or system-level software) or learning about the strengths and weaknesses of the opponent (*e.g*., that the opponent uses non-conventional means to cause a buffer overflow, indicating that it is likely that the opponent was created by a technology-savvy terrorist or information warrior sponsored by a nation-state), either terminates the interaction (*e.g.*, closing a communication port) or mounts an information warfare campaign against the opponent, at which point the opponent is treated as a *cyber combatant*. Here we define a cyber combatant to be a proxy for an information warrior who has the rights under international law to participate directly in armed conflict during hostilities.

The formalism proposed by Michael *et al*. [5] for specifying rules for runtime intrusion detection and corresponding countermeasures is based on behavior patterns over event traces and a catalog of decoy actions, such as blocking or substituting certain responses of applications, middleware, or system commands.

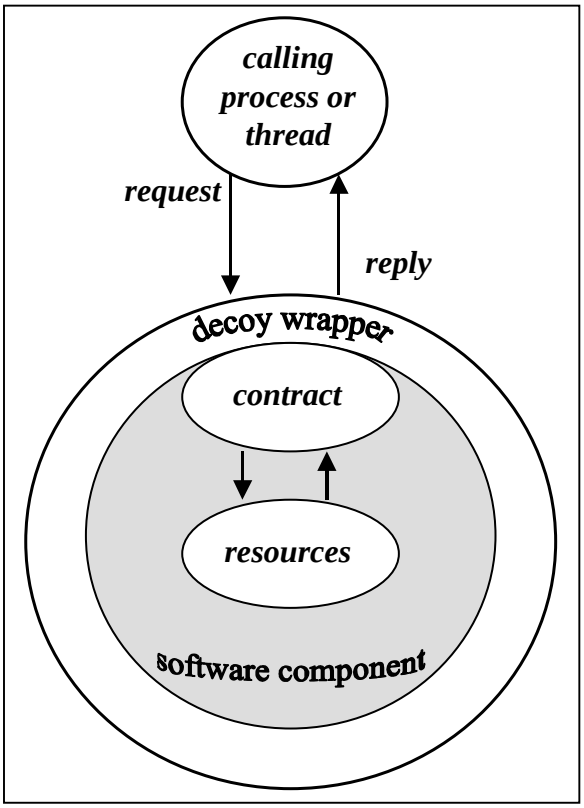

Figure 1. A decoy-enabled atomic software component

The implementation of the formalism is based on automatic instrumentation for event detection derived from the behavior model of the system to be defended against intrusions, supporting the rapid wrapping of software components as the software decoys learn about the nature and source of attacks. Software components are wrapped with decoy functionality on a selective basis; wrapping can be performed at more than one level of abstraction, from application-level objects such as web applets to low-level operating system calls. If any of the assertions (*e.g*., preconditions, postconditions, class invariants) built into the component's contract are violated, the software decoy transfers the interaction with the opponent to an *antechamber* [6] that is akin to a virtual sandbox; the antechamber is either hosted by the operational system on which the software decoy resides, or on a separate processor or platform in order to minimize the affect of the monitoring and decoy actions (*e.g*., those of delay tactics) on the availability and performance of computing resources requested by legitimate users of the decoy-enabled software components.

All of the decoying actions are performed inside the antechamber in order to protect the software component and the system on which the component resides from the effects of malicious requests for service. For instance, a rogue agent could pass a character string that exceeds the maximum buffer size allocated by a module for such strings, with the intent to trick the module into executing a Trojan horse when an exception is raised due to the resulting buffer overflow. The software decoy will transfer the call to the antechamber of the software component, in which the call will be allowed to execute and the component will respond to the calling process or thread using the detection and deception rules with which the component was instrumented.

Intrusion detection rules are textually separated from the source code of the system, which allows for accumulating and formalizing knowledge of typical intrusion patterns and decoying strategies. The formalism provides for specifying a broad spectrum of decoying strategies, from simple delaying tactics to the simulation of the behavior of real applications, middleware, or operating systems.

3. *Jus in Bello* Applied to Software Decoys

There is some risk that enacting the response policy of a software decoy could in some way violate the rules of armed conflict, known as *jus in bello*. Wingfield [10] has laid out a framework from which to evaluate whether various acts taken as part of information warfare or information operations are compliant with the rules of war. The four customary guiding principles used in such an evaluation are discrimination, necessity, proportionality, and chivalry. Another dimension to be considered is whether the interaction between two opposing parties is in the stage of pre-hostilities or hostilities. We address each of the guiding principles with respect to intelligent software decoys, using the rogue-agent scenario given in Section 1 as the motivating example.

3.1. Discrimination

Suppose that the software decoy identifies patterns of behavior exhibited by the software opponent that indicate that the opponent is trying to modify the behavior of the software component such that the component will corrupt the integrity of the XML tags. What response can the software decoy provide to the opponent? First, the software decoy must answer the following question: Can the software decoy treat the opponent as a cyber combatant? The answer is yes, according to the principle of discrimination, only if the software decoy can determine that the opponent has the rights under international law to participate directly in armed conflict during hostilities.

Further, suppose that the software decoy can determine that the opponent is a combatant through various means, such as by recognizing a distinguishing pattern of attack or the origination point of the attack known to be that of a hostile military force. If the decoy has decided to discontinue generating decoying actions and now move to reduce or eliminate the ability of the combatant to attack, can the software decoy launch a counterattack on the combatant? The principle of discrimination limits the response of the software decoy to be applied to the cyber opponent. In response to the provocation, a software decoy cannot use cyber weapons that would have an indiscriminate effect. For instance, if the origin of the attack happens to be an information infrastructure that is used by both combatants and noncombatants alike, the attack response of the decoy must be directed against a specific military objective on the target infrastructure. This would preclude the software decoy from, for instance, launching a computer worm that could not precisely target the military objective without harming the software components of noncombatants on the shared infrastructure. (*N.B.*: The U.S. military along with many of its foreign counterparts rely to a large extent on the civilian information infrastructure for computing and communications.)

Now suppose the software decoy does have precision-guided cyber weapons at its disposal. Can these weapons be used against the cyber combatant? From a technical perspective the answer may be no because if a combatant's software component is damaged or destroyed, this may result in harmful side effects on civilian software components, such as the damaged military component altering the state of an object such that the object exports defective or inappropriate features to civilian components, causing those civilian components to experience faults or failures. The cyber opponent may intentionally build such side effects into software components with the aim of deterring counterattacks.

The foregoing discussion applies to decoying responses too. For example, instead of counterattacking or terminating interaction with the cyber combatant, a decoy could continue to try to lessen the willingness of the combatant to attack by returning a faked but believable result (*e.g.*, part of an ontology used by the targeted semantic web) to a process or thread, causing the receiving military software component to generate side effects (*e.g.*, a buffer overflow) in a manner that harms civilian components. One can contend that the indiscriminate use of side effects, either by the software decoy or cyber combatant, would fail the litmus test for discrimination.

3.2. Necessity

The principle of necessity is used to determine what amount of force a software decoy can employ against a cyber combatant. For example, it would be unlawful for the software decoy to launch a cyber weapon of mass destruction such as a lethal computer virus, or create an illu-

sion through deception, either of which would have an effect beyond that required for accomplishing the military mission at hand. The software decoy can be preprogrammed with only lawful means and methods for responding to provocation. A software decoy is an inanimate object, so it has no will of its own—it does what it is programmed to do by the human information warrior.

In addition to the quantitative aspect of necessity (*i.e.*, assessing the amount of force to be applied), there is a qualitative aspect, that being the assessment of the type of force to be applied. Certain types of cyber weapons will be *per se* out of bounds for use in conflicts, those being the analogues of for example biological weapons and transparent bullets. For instance, it would be unlawful for a software decoy to launch remote method invocations that instruct an adversary's software components, used for controlling the production of industrial chemicals, to release those chemicals (*e.g.*, pesticides produced near a population center or military installation) into the atmosphere.

3.3. Proportionality

The intelligence-gathering capability of software decoys makes it possible for decoys to determine the nature, magnitude, severity, and source of an attack. For example, cooperating software decoys may determine that their present interaction with cyber opponents will, with high probability, have a crippling impact on the cooperative engagement grid the software decoys are suppose to protect. Given these circumstances, what should be the response of the software decoy to the provocation? The principle of proportionality provides guidance here. The decoy should not naively apply the same amount of force in return, nor should it necessarily use the same type of cyber weapons or targets as those used by the cyber combatant. Instead, the software decoy must "take all reasonable precautions" in selecting the deception or counterattack response such that the response is not excessive in relation to the concrete and direct military advantage anticipated by the software decoy.

An issue here is can a military force delegate to the software decoy the authority to judge whether all feasible precautions have been taken, or what constitutes excessive force? How would the software decoy determine when it has collected and analyzed a sufficient amount of reliable and accurate intelligence data to know the true nature of the target of its deception or counterattack? It may turn out that certain types of responses generated by software decoys will require approval of a *cyber military commander* that is akin to an oracle that has situational awareness of the battlespace, which in term would need approval from a human military commander to provide the response. Such a chain of command might hamper the ability of the software decoy to generate effective deceptions due to the delays inherent in obtaining approval through the chain of command.

Another issue is the following: How would a software decoy determine with certainty that a response directed at a military object will not have unintended consequences? Software tends to be complex and large, making it difficult to test for the existence of such effects and software bugs in general. In addition, there are limitations on the time in which the software decoy must act before the cyber combatant responds to the decoying actions. Testing takes time and may not even be possible if the software decoy or other decoys with which it cooperates cannot infiltrate the adversary's rogue agents or information infrastructure. Lastly, the passing of object code, such as is done in runtime extensible virtual environments, makes it even more difficult to test for side effects.

Regarding the legal standard applied by the United States of "taking all reasonable precautions," when software decoys are used in a cooperative engagement grid by allied or coalition forces, the decoys will be held to higher standards such as that of the North Atlantic Treaty Organization (NATO), which is "take all feasible precautions." Thus, a software decoy will need to have the ability to switch between sets of decoy-and-attack responses corresponding to different legal standards as the composition of the allied or coalition forces change.

Designing software decoys to adhere to the Unified Power of Attack is a way forward for minimizing the risk of violating the principle of proportionality. In the first two stages of attack, the software decoy deflects rather than tries to harm the cyber opponent. Force is applied against the opponent only as a means of self-defense and option of last resort.

3.4. Chivalry

Software decoys rely on ruses of war to deceive an opponent. For example, in response to the rogue agent that is tasked with modifying the XML tags, the software decoy could try to deflect the cyber opponent by redirecting the attention of the opponent to a honeypot that contains faked XML tags that to the opponent appear to have greater strategic or tactical value than those managed by the software component it is currently attacking.

According to the principle of chivalry, ruses of war cannot be used to kill, injure, or capture an adversary by resort to perfidy. An example of a perfidious act by a software decoy would be to feign a noncombatant status during hostilities; this act is perfidious because it invites the confidence of the cyber opponent into believing that it is obligated to accord the software decoy protection as a noncombatant, with intent to betray that confidence. The danger here is that once that confidence is betrayed, the opponent may attack real noncombatant software compo-

nents in the belief that those components are military targets masquerading as noncombatants.

During pre-hostilities, the software decoy can legally take on a noncombatant appearance. It can also do so while using deception to either reduce or eliminate the cyber combatant's will to attack or to change the proximity of itself to the combatant. However, before the software decoy can apply force (*i.e.*, to reduce or eliminate the ability of the combatant), the decoy must show its true colors (*i.e.*, identify itself as combatant).

However, a software decoy used by a combatant can never legally take on a protected appearance, for example, that of a software component belonging to a neutral state or state not party to the conflict, as this would endanger software components that really should be accorded protected status.

One issue that immediately comes to mind is what does it mean for a cyber opponent to have misplaced belief or confidence in something? Goldberg [11] and Hirstein [12] have pondered the issue of whether software components can possess conflicting representations that could result in a form of software-based self-deception.

Another issue is what defines the appearance of a software decoy? Is it the location of a software decoy within an information grid? Is it defined by the specification of the contract for a software decoy? Or might it be a combination of these and other attributes of the software decoy, including the observation by an opponent of a decoy-like behavior exhibited by the a software component?

Another level of complexity is introduced here by the fact that software decoys can disguise themselves by altering the appearance of software components. This chameleon-like ability is supported by the use of polymorphic types; polymorphism permits late binding of the message interaction. For example, a software decoy can dynamically change the number of arguments, the order of arguments, or the data type or class of arguments of its interface signature (*i.e.*, the contract that is advertised to other components).

4. Management of Deception Policy

Based on the foregoing discussion, we believe there is a need to incorporate some means of managing the deception policy governing the responses that may be employed by software decoys. If a software decoy contravenes any of the four guiding principles, then it could be deemed to be a cyber war criminal. Ultimately, however, the information warrior and others higher up in the chain of command will be held legally culpable for the actions of the decoy for two reasons: (i) to reiterate, the intent of the information warrior is embedded in the deception and attack responses of the software decoy and (ii) the commanders cannot delegate their authority in order to release themselves from responsibility for the acts of their subordinates—including the actions of decoys.

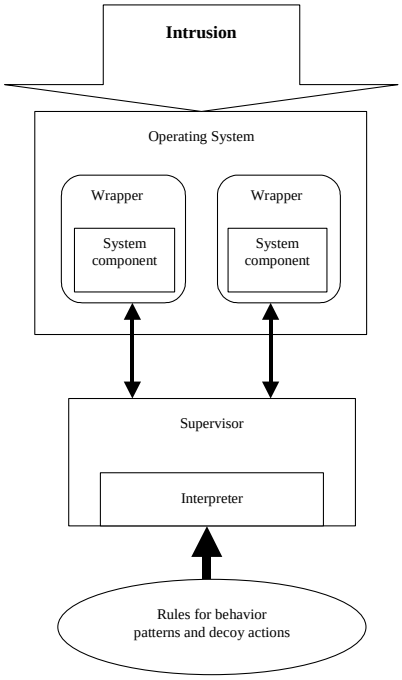

Figure 2. Software-decoy architecture (from [5])

The *supervisor* shown in Figure 2 manages deception policy. In this architecture, the supervisor coordinates the exchange of information among software decoys and their actions, although the software decoys are designed to be able to operate in an autonomous manner when communication with the supervisor is not possible or ill advised. It also instruments software decoys with behavior patterns for use in detecting malicious behavior of calling threads and processes, along with decoy actions to be taken when those patterns of behavior are detected.

One advantage of such a management scheme is that a military force, via the supervisor, could rapidly reprogram the decoy defenses from that of a pre-hostilities mode to a hostilities mode. In the pre-hostilities mode, certain types of responses would not be available to the software decoy, such as the use of potentially dangerous deception strategies or cyber weapons of last resort. This approach would remove to a large extent the need for the software decoy to reason about the consequences of its actions in terms of *jus in bello* and instead focus on applying tactics to achieve effective deceptions against their opponents.

In contrast, the supervisor would be responsible for reasoning about both the potential effects of applying specific sets of decoying strategies in various contexts of conflict and the strategy for applying the decoy responses. In this sense, the supervisor plays the role of a cyber military commander (*viz.* Section 3.3). An integrated set

of computer-based tools, such as those identified in what has been termed a "policy workbench" [13], would assist the supervisor in specifying, reasoning about, maintaining, and enforcing doctrine and policy about deception.

Another advantage of this management scheme is that doctrine and policy developed for software decoys could be integrated into a larger joint information operations and warfare doctrine-policy base and be distributed to the supervisors throughout the cooperative engagement grid. It could also serve as a reference for automatically applying different legal standards as they pertain to information operations and warfare among allied and coalition forces. However, care should be taken to not overly standardize deception responses. As pointed out by Fowler and Nesbit [14], "the most effective deception will be imaginative and creative; it cannot be 'legislated' or 'ordered'; and it must not become stereotyped or 'bureaucratized.'"

5. Conclusion

There are many open issues to be explored for evaluating the appropriateness of responses of intelligent software decoys to provocation in the context of *jus in bello*, in addition to the locus for making such decisions (*i.e.*, locally by the software decoy or globally by a supervisor). Resolving such issues in the context of semantic webs that will support cooperative engagement grids will require consideration of the specific requirements and characteristics of such grids, such as the fact that software components comprising the web or grid will likely need to process increasingly larger volumes of transactions as the intensity of a conflict grows and there is a corresponding need for more rapid and efficient sharing of knowledge about the battlespace, with any of the interactions involved in those transactions potentially requiring active responses from software decoys.

6. Acknowledgements

I thank Thomas Wingfield of Aegis Corporation for reviewing this manuscript and giving such a captivating lecture, titled "How Not to be a War Criminal in Cyberspace: 4 Easy Steps," at the Phoenix Challenge 2002 Conference (San Antonio, Texas, 23 April 2002); that lecture inspired me to try my hand at applying the guiding principles of *jus in bello* to a software abstraction that I think could shape the future of information operations and warfare—the intelligent software decoy. I also thank Gregory Larsen of the Institute for Defense Analyses for clarifying for me his conceptualization of knowledge-centric warfare. This research is supported by the Naval Research Laboratory under contract no. N41756-01-WR-10433. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotations thereon.

References

[1] Cebrowski, A. K. and Garstka, J. J. Network-centric warfare: Its origin and future. *U.S. Naval Inst. Proc*. 124, 1 (Jan. 1998), 28-35.

[2] Larsen, G. N. Institute for Defense Analyses, Alexandria, Va. Interview, 18 Sept. 2001.

[3] Berners-Lee, T., Hendler, J., and Lassila, O. The semantic web. *Sci. Amer*. 284, 5 (May 2001), 34-43.

[4] Thuraisingham, B., Hughes, E., and Allen, D. Dependable semantic web. In *Proc. Seventh Int. Workshop Object-oriented Real-Time Dependable Syst.*, IEEE (San Diego, Calif., Jan. 2002), 305-308.

[5] Michael, J. B., Auguston, M., Rowe, N. C., and Riehle, R. D. Software decoys: Intrusion detection and countermeasures. In *Proc. Workshop on Inf. Assurance*, IEEE (West Point, N.Y., June 2002).

[6] Michael, J. B. and Riehle, R. D. Intelligent software decoys. In *Proc. Monterey Workshop: Eng. Automation for Software Intensive Syst. Integration*, n.p., (Monterey, Calif., June 2001), 178-187.

[7] Szyperski, C. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley: Harlow, Eng., 1999.

[8] Meyer, B. *Object-Oriented Software Construction*. Prentice-Hall: Upper Saddle River, N.J., 1998.

[9] Westbrook, A. and Ratti, O. *Aikido and the Dynamic Sphere*. Rutland, Vt.: Charles E. Tuttle Co., 1970.

[10] Wingfield, T. C. *The Law of Information Conflict: National Security Law in Cyberspace*. Falls Church, Va.: Aegis Research Corp., 2000.

[11] Goldberg, S. C. The very idea of computer self-knowledge and self-deception. *Minds and Machines* 7, 4 (Nov. 1997), 515-529.

[12] Hirstein, W. Self-deception and confabulation. *J. Phil. Sci.* 67, 3 (Suppl. S, Sept. 2000), S418-S429.

[13] Sibley, E. H., Michael, J. B., and Wexelblat, R. L. Use of an experimental policy workbench: Description and preliminary results. *IFIP Trans. A*-6 (1992), 47-76.

[14] Fowler, C. A. and Nesbit, R. F. Tactical deception in air-land warfare. *J. Electronic Defense* 18, 6 (June 1995), 37-44 and 76-79.